




Full Stack Development



UNIT III

Full Stack Development


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Learning Resources

- Books
 - D. Brad, B. Dayley and C. Dayley, "Node.js, MongoDB and Angular Web Development: The definitive guide to using the MEAN stack to build web applications", Addison-Wesley Professional, 2nd Edition, 2017
- Web Links (Strictly Referred):
 - <https://angular.io/>
 - <https://nodejs.org/>
 - <https://expressjs.com>

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Course Outcome

- CO1: Relate the basics of Javascript (JS) and ReactJS
- CO2: Apply the concepts of props and State Management in React JS
- CO3: Examine Redux and Router with React JS
- CO4: Appraise Node JS environment and modular development
- CO5: Develop full stack applications using MongoDB

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Overview

UNIT-3

- Introduction to Angular
 - Angular architecture; introduction to components, component interaction and styles; templates, interpolation and directives; forms, user input, form validations; data binding and pipes; retrieving data using HTTP; Angular modules
- Node.js
 - Introduction, Features, Node.js Process Model
 - Environment Setup: Local Environment Setup, The Node.js Runtime, Installation of Node.js
 - Node.js Modules: Functions, Buffer, Module, Modules Types
 - Node Package Manager: Installing Modules using NPM, Global vs Local Installation, Attributes of Package.js on, Updating packages, Mobile-first paradigm, Using twitter bootstrap on the notes application, Flexbox and CSS Grids
 - File System: Synchronous vs Asynchronous, File operations
 - Web Module: Creating Web Server, Web Application Architecture, Sending Requests, Handling http requests
 - Express Framework: Overview, Installing Express, Request / Response Method, Cookies Management

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Angular

- Angular is a platform and framework for building single-page client applications using HTML and TypeScript
 - TypeScript is JavaScript with syntax for types
 - TypeScript adds additional syntax to JavaScript to support a **tighter integration** with your editor. Catch errors early in your editor.
 - TypeScript code converts to JavaScript, which **runs anywhere JavaScript runs**: in a browser, on Node.js

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Angular

- The main building blocks of an Angular application:
 - **Modules**: modules are highly recommended because they allow you to separate your code into separate files
 - **Data binding**: the process of **linking data from a component with** what is displayed in a web page
 - **Services**: Services are **singleton classes** that provide functionality for a web app. The service functionality is **completely independent of context or state**, so it can be easily consumed from the components of an application
 - **Dependency injection**: a process in which a **component defines dependencies** on other components. When the code is **initialized**, the **dependent component is made available for access within the component**

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Angular

- The Eight main building blocks of an Angular application:
 - **Directives:** Directives are JavaScript classes with metadata that defines the structure and behavior
 - **Components:** A component directive is a directive that incorporates an HTML template with JavaScript functionality to create a self-contained UI element
 - **Structural:** You use structural directives when you need to manipulate the DOM
 - **Attribute:** An attribute directive changes the appearance and behavior of HTML elements by using HTML attributes

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Angular

- Components are the main building block for Angular applications. Each component consists of:
 - An HTML template that declares what renders on the page
 - A TypeScript class that defines behavior
 - A CSS selector that defines how the component is used in a template
 - Optionally, CSS styles applied to the template
- Creating a component
 - To create a new component manually:
 - Navigate to your Angular project directory.
 - Create a new file, <component-name>.component.ts.
 - At the top of the file, add the following import statement.
 - `import { Component } from '@angular/core';`

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Angular

- Install angular
 - `npm install -g @angular/cli`
- Create a new App
 - Syntax: `ng new <app-name>`
 - Example: `ng new test01`
- Run Angular Project (inside the project folder)
 - Syntax: `ng serve --open`
- Files:
 - `app.component.ts` : The component class code, written in TypeScript
 - `app.component.html`: The component template, written in HTML
 - `app.component.css`: The component's private CSS styles.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

```

FORM-BUILDING
└─> node_modules
├─> src
│   ├── form-demo
│   ├── form-demo.component.html
│   ├── form-demo.component.ts
│   ├── app-components.html
│   ├── app-components.ts
│   ├── app.module.ts
│   ├── app.ts
│   └── assets
└─> environments
├─> browser
└─> polyfills

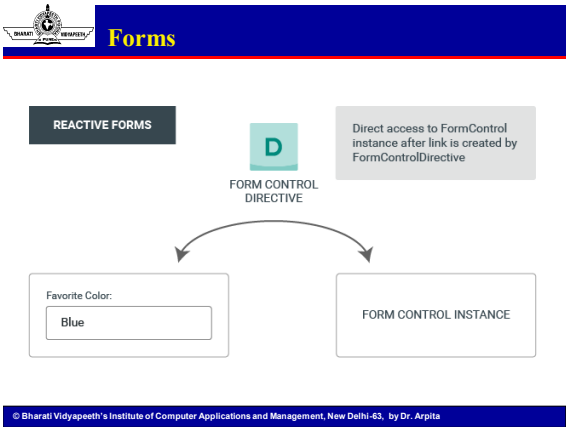
```

```

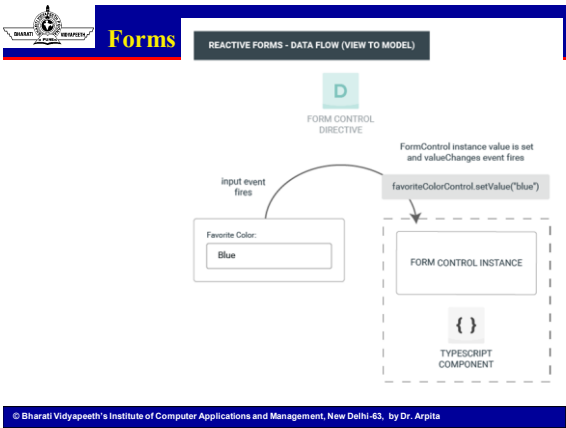
1 <div style="text-align: center;">
2   <form #login="login" (ngSubmit) = "submit(login)">
3     <div class="form-group">
4       <label>Email address</label>
5       <input #email="email" #maxlength="30" required type="email" class="form-control" placeholder="Enter email" ngModel />
6       <div *ngIf="$form.touched && $form.invalid" class="alert alert-danger">
7         <div>
8           Email is required
9         </div>
10      </div>
11    <div *ngIf="$form.errors?.maxlength">
12      Email is invalid
13    </div>
14  </div>
15  <div class="form-group">
16    <label>Password</label>
17    <input type="password" class="form-control" id="exampleInputPassword" placeholder="Password" ngModel />
18  </div>
19  <button type="submit" class="btn btn-primary">Submit</button>
20 </form>

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

Forms REACTIVE FORMS - DATA FLOW (MODEL TO VIEW)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

Forms

- Reactive form data flow View-to-model

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

Forms

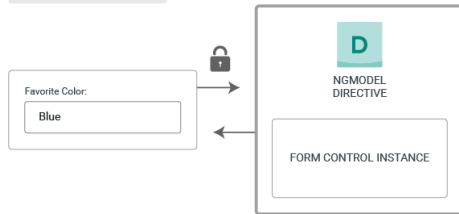
- Reactive form data flow model-to-view

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

Forms

Can only access
FormControl instance via
NgModel directive

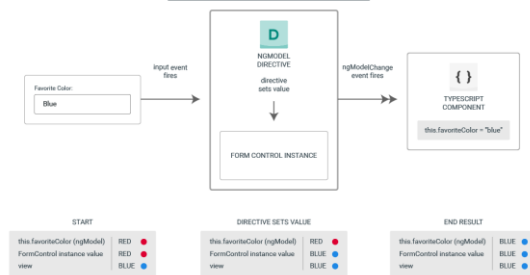
TEMPLATE-DRIVEN FORMS



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

Forms

TEMPLATE-DRIVEN FORMS - DATA FLOW (VIEW TO MODEL)



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

User Input

- User actions such as clicking a link, pushing a button, and entering text raise DOM events
 - Binding to user input events

```
@Component({
  selector: 'app-click-me',
  template: `
    <button type="button" [click]="onClickMe()">Click me!</button>
    {{clickMessage}}`
})
export class ClickMeComponent {
  clickMessage = '';

  onClickMe() {
    this.clickMessage = 'You are my hero!';
  }
}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



User Input

- User actions such as clicking a link, pushing a button, and entering text raise DOM events
 - Get user input from the Sevent object

```

template: `
  <input (keyup)="onKey($event)">
  <p>{{values}}</p>
`
export class KeyUpComponent_v1 {
  values = '';

  onKey(event: any) { // without type info
    this.values += event.target.value + ' | ';
  }
}

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



User Input

- User actions such as clicking a link, pushing a button, and entering text raise DOM events
 - Type the Sevent

```

export class KeyUpComponent_v1 {
  values = '';

  onKey(event: KeyboardEvent) { // with type info
    this.values += (event.target as HTMLInputElement).value + ' | ';
  }
}

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



User Input

- User actions such as clicking a link, pushing a button, and entering text raise DOM events
 - Get user input from a template reference variable

```

@Component({
  selector: 'app-loop-back',
  template: `
    <input #box (keyup)="0">
    <p>{{box.value}}</p>
  `
})
export class LoopbackComponent { }

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Form Validation

- Validating input in reactive forms

VALIDATOR
TYPE

DETAILS

Sync validators Synchronous functions that **take a control instance and immediately return either a set of validation errors or null**. Pass these in as the second argument when you instantiate a FormControl.

Async validators Asynchronous functions that **take a control instance and return a Promise or Observable that later emits a set of validation errors or null**. Pass these in as the third argument when you instantiate a FormControl.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Form Validation (Built-in validator functions)

```
ngOnInit(): void {
  this.heroForm = new FormGroup({
    name: new FormControl(this.hero.name, [
      Validators.required,
      Validators.minLength(4),
      forbiddenNameValidator(/bob/i) // <-- Here's how you pass in the custom
    validator.
  ]),
    alterEgo: new FormControl(this.hero.alterEgo),
    power: new FormControl(this.hero.power, Validators.required)
  });
}

get name() { return this.heroForm.get('name'); }
get power() { return this.heroForm.get('power'); }
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Form Validation (Defining custom validators)

```
export function forbiddenNameValidator(nameRe: RegExp): ValidatorFn {
  return (control: AbstractControl): ValidationErrors | null => {
    const forbidden = nameRe.test(control.value);
    return forbidden ? {forbiddenName: {value: control.value}} : null;
  };
}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Adding cross-validation to template-driven forms

- For a template-driven form, you must create a directive to wrap the validator function. You provide that directive as the validator using the NG_VALIDATORS token.

```
@Directive({
  selector: '[appIdentityRevealed]',
  providers: [{ provide: NG_VALIDATORS, useExisting:
    IdentityRevealedValidatorDirective, multi: true }]
})
export class IdentityRevealedValidatorDirective implements Validator {
  validate(control: AbstractControl): ValidationErrors | null {
    return identityRevealedValidator(control);
  }
}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Asynchronous validators

- Asynchronous validators implement the AsyncValidatorFn and AsyncValidator interfaces.
- Differ than synchronous
 - The `validate()` functions must return a Promise or an observable,
 - The **observable returned must be finite**, meaning it must complete at some point. To **convert an infinite observable into a finite one**, pipe the observable through a filtering operator such as `first`, `last`, `take`, or `takeUntil`.
- Asynchronous validation is performed **after the synchronous validation and only if the synchronous validation is successful**.
- After asynchronous validation begins, the form control enters a pending state. Inspect the control's `pending` property and use it to give visual feedback about the ongoing validation operation.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



synchronous validators works

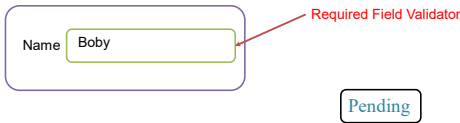
Name → Required Field Validator

Synchronous
Field
Validator

~~Asynchronous
Field
Validator~~

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

synchronous validators works



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

Adding async validators to template-driven forms

```

@Directive({
  selector: '[appUniqueAlterEgo]',
  providers: [
    {
      provide: NG_ASYNC_VALIDATORS,
      useExisting: forwardRef(() => UniqueAlterEgoValidatorDirective),
      multi: true
    }
  ]
})
export class UniqueAlterEgoValidatorDirective implements AsyncValidator {
  constructor(private validator: UniqueAlterEgoValidator) {}

  validate(
    control: AbstractControl
  ): Observable<ValidationErrors | null> {
    return this.validator.validate(control);
  }
}

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

Data Binding & Pipes

- Data binding automatically keeps your page up-to-date based on your application's state in Angular.
- Data binding works with properties of DOM elements, components, and directives, not HTML attributes
- Attributes initialize DOM properties and you can configure them to modify an element's behavior

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Binding types and targets

- Type: Property
- Target:
 - Element property
 - Component property
 - Directive property

```
<img [alt]="hero.name" [src]="heroImageUrl">
<app-hero-detail [hero]="currentHero"></app-
hero-detail>
<div [ngClass]="{'special': isSpecial}"></div>
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Binding types and targets

- Type: Event
- Target:
 - Element event
 - Component event
 - Directive event

```
<button type="button"
(click)="onSave()">Save</button>
<app-hero-detail
(deleteRequest)="deleteHero()"></app-hero-
detail>
<div (myClick)="clicked=Sevent" clickable>click
me</div>
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Binding types and targets

- Type: Attribute `<button type="button" [attr.aria-label]="help">help</button>`
- Target:
 - Attribute
- Type: Class
- Target: `<div [class.special]="isSpecial">Special</div>`
 - class property

- Type: Style
- Target: `<button type="button" [style.color]="isSpecial ? 'red' : 'green'">`
 - style property

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Pipe

- Decorator that marks a class as pipe and supplies configuration metadata.
- Angular Pipes allows its users to change the format in which data is being displayed on the screen.



- Angular pipes are simple functions designed to accept an input value, process, and return a transformed value as the output.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Pipe

- Pipes are defined using the pipe “|” symbol.
- Pipes can be chained with other pipes.
- Pipes can be provided with arguments by using the colon (:) sign.
- Types of Pipes (in-built)
 - DatePipe: Formats a date value.
 - UpperCasePipe: Transforms text to uppercase.
 - LowerCasePipe: Transforms text to lowercase.
 - CurrencyPipe: Transforms a number to the currency string.
 - PercentPipe: Transforms a number to the percentage string.
 - DecimalPipe: Transforms a number into a decimal point string.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Pipe.component.ts

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-pipes',
  templateUrl: './pipes.component.html',
  styleUrls: ['./pipes.component.css']
})
export class PipesComponent implements OnInit {
  dateToday: string;
  name: string;
  constructor() { }
  ngOnInit(): void {
    this.dateToday = new Date().toLocaleDateString();
    this.name = "BVICAM"
  }
}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Pipe.component.html

```

<h1>
Date: {{ dateToday }} <br>
Date Pipe: {{ dateToday | date | uppercase }}<br>
Date Pipe: {{ dateToday | date: 'short' | lowercase }} <br>
Name: {{ name | uppercase }} <br>
Name: {{ name | slice:6 }}

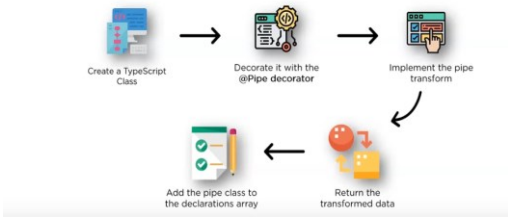
</h1>

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Creating Custom Pipes



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Creating Custom Pipes



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Getting data from backend

- Enable HTTP services
 - HttpClient is Angular's mechanism for communicating with a remote server over HTTP.
 - Make HttpClient available everywhere in the application in two steps. First, add it to the root AppModule by importing it:

```
src/app/app.module.ts (HttpClientModule import)
```

```
import { HttpClientModule } from '@angular/common/http';
```

- Next, still in the AppModule, add HttpClientModule to the imports array:

```
src/app/app.module.ts (imports array excerpt)
```

```
@NgModule({
  imports: [
    HttpClientModule,
  ],
})
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Angular Fetch Data from API Using HttpClientModule

- Step 1: Add HttpClientModule Into the Imports Array and Import it.
- Step 2: Create an Instance of HttpClient and Fetch the Data Using It
- Step 3: Create a Student Interface To Cast the Observables
- Step 4: Subscribe the Data From the Service in the Component

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Angular Modules

- Angular applications are modular and Angular has its own modularity system called NgModules.
- NgModules are containers for a cohesive block of code dedicated to
 - an application domain,
 - a workflow, or
 - a closely related set of capabilities
- They can contain
 - components,
 - service providers, and
 - other code files whose scope is defined by the containing NgModule.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Angular Modules

- An NgModule is defined by a class decorated with @NgModule().
- The @NgModule() decorator is a function that takes a single metadata object, whose properties describe the module
- Properties are:
 - declarations: -The components, directives, and pipes that belong to this NgModule.
 - exports: The subset of declarations that should be visible and usable in the component templates of other NgModules.
 - imports: Other modules whose exported classes are needed by component templates declared in this NgModule.
 - providers: Creators of services that this NgModule contributes to the global collection of services
 - bootstrap: The main application view, called the root component, which hosts all other application views.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Example of simple root module (app.module.ts)

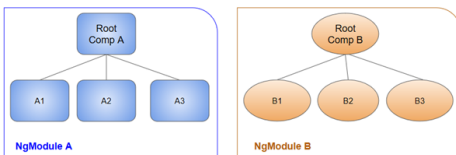
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports: [ BrowserModule ],
  providers: [ Logger ],
  declarations: [ AppComponent ],
  exports: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



NgModules and components

- NgModules provide a compilation context for their components.
- A root NgModule always has a root component that is created during bootstrap
- Any NgModule can include any number of additional components, which can be loaded through the router or created through the template

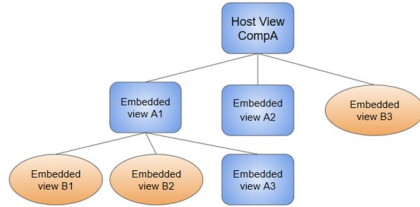


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



NgModules and components

- A component and its template together define a view. A component can contain a view hierarchy, which allows you to define arbitrarily complex areas of the screen that can be created, modified, and destroyed as a unit.



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node JS

- Node.js is an open-source and cross-platform JavaScript runtime environment.
- Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant.
- A Node.js app runs in a single process, without creating a new thread for every request
- Node.js provides a set of asynchronous I/O primitives in its standard library
- When Node.js performs an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread, Node.js will resume the operations when the response comes back


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node JS

- Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency
- npm with its simple structure helped the ecosystem of Node.js proliferate, and now the npm registry hosts over 1,000,000 open source packages you can freely use

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node JS

```

indexjs x
const http = require('http')


const hostname = '127.0.0.1'
const port = 3000

const server = http.createServer((req, res) => {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/plain')
  res.end('Hello World\n')
})

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`)
})

```


© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



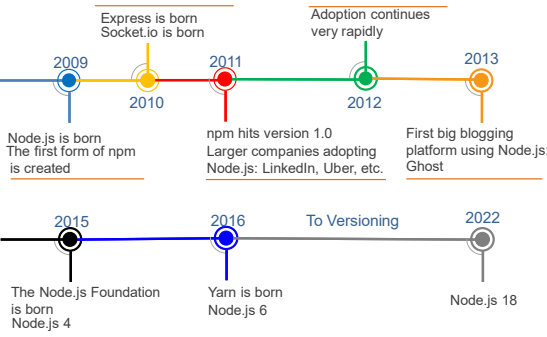
Node.js Frameworks and Tools

- AdonisJS: A TypeScript-based fully featured framework highly focused on developer ergonomics, stability, and confidence. Adonis is one of the fastest Node.js web frameworks.
- Express: It provides one of the most simple yet powerful ways to create a web server. Its minimalist approach, unopinionated, focused on the core features of a server, is key to its success.
- hapi: A rich framework for building applications and services that enables developers to focus on writing reusable application logic instead of spending time building infrastructure.
- Loopback.io: Makes it easy to build modern applications that require complex integrations.
- Micro: It provides a very lightweight server to create asynchronous HTTP microservices.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



History of Node JS



The timeline shows key milestones in Node.js development:

- 2009:** Node.js is born. The first form of npm is created.
- 2010:** Express is born, Socket.io is born.
- 2011:** npm hits version 1.0. Larger companies adopting Node.js: LinkedIn, Uber, etc.
- 2012:** Adoption continues very rapidly.
- 2013:** First big blogging platform using Node.js: Ghost.
- 2015:** The Node.js Foundation is born. Node.js 4.
- 2016:** Yarn is born. Node.js 6.
- 2022:** Node.js 18.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



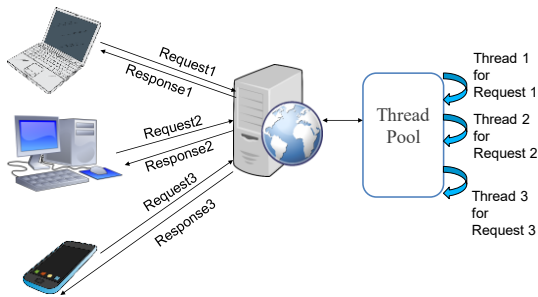
Node.js Frameworks and Tools

- Cross-platform compatibility
- Asynchronous and Event Driven
- Single Threaded but Highly Scalable
- The convenience of using one coding language
- V8 Engine
- Facilitates quick deployment and microservice development
- No Buffering
- Commendable data processing ability
- Active open-source community
- Additional functionality of NPM
- Advanced hosting ability of NodeJs

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



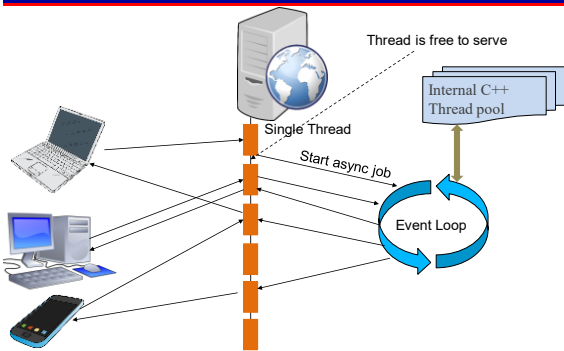
Conventional Process Model



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node JS Process Model



© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Conventional Process Model

- Cross-platform compatibility
- Asynchronous and Event Driven
- Single Threaded but Highly Scalable
- The convenience of using one coding language
- V8 Engine
- Facilitates quick deployment and microservice development
- No Buffering
- Commendable data processing ability
- Active open-source community
- Additional functionality of NPM
- Advanced hosting ability of NodeJs

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node Modules

- In simple terms, a module is code that we group together for the purposes of sharing and reuse
- Modules, therefore, allow us to break down complexity in our applications into small chunks
- In Node.js, each JavaScript file as a separate module.
- When Node.js was invented, modules and code encapsulation were high on the priority list

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Module.exports

- module.exports are part of the CommonJS specification
- Module exports are the instruction that tells Node.js which bits of code (functions, objects, strings, etc.) to “export” from a given file so other files are allowed to access the exported code
- Make it more clear:
 - Suppose all the modules written together in single file.
 - The code would look like each module wrapped in a function and given an argument, which is the current module.

```
(function(exports, require, module, __filename, __dirname) {
  // Module code actually lives in here
});
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Exporting Module

```

module.exports.getUser = () => {
  // Code here
}

module.exports.getUsers = () => {
  // Code here
}

function getUser() {
  // Code here
}

function getUsers() {
  // Code here
}

module.exports = {
  getUser,
  getUsers
}

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Cycle

- When there are circular require() calls, a module might not have finished executing when it is returned.

```

a.js:          b.js:
console.log('a starting'); console.log('b starting');
exports.done = false;     exports.done = false;
const b = require('./b.js'); const a = require('./a.js');
console.log('in a, b.done = %j', b.done); console.log('in b, a.done = %j', a.done);
exports.done = true;     exports.done = true;
console.log('a done');    console.log('b done');

main.js: console.log('main starting');
const a = require('./a.js');
const b = require('./b.js');
console.log('in main, a.done = %j, b.done = %j', a.done, b.done);

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Modules

- If the exact filename is not found, then Node.js will attempt to load the required filename with the added extensions: .js, .json, and finally .node.
- There are three ways in which a folder may be passed to require() as an argument.
 - The first is to create a package.json file in the root of the folder, which specifies a main module.


```

{ "name": "some-library",
  "main": "./lib/some-library.js" }

```
- Loading from node_modules folders
- Loading from the global folders
 - Get a path from ENV Variables
- The module wrapper


```

(function(exports, require, module, __filename, __dirname) {
  // Module code actually lives in here
});

```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Buffer

- The buffer module from node.js, for the browser.
- Buffer objects are used to represent a fixed-length sequence of bytes.
- Super fast.
- Extremely small bundle size (6.75kb)
- Excellent browser support
- Square-bracket buf[4] notation works!
- Install buffer: `npm i buffer`

```
var Buffer = require('buffer/').Buffer
const buf = Buffer.from([1, 2, 3, 4]);
const uint32array = new Uint32Array(buf);
console.log(uint32array);
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node Package Manager (npm)

- npm is the standard package manager for Node.js.
- It started as a way to download and manage dependencies of Node.js packages, but it has since become a tool used also in frontend JavaScript.
- Yarn and pnpm are alternatives to npm cli. You can check them out as well.
- Installing all dependencies
 - If a project has a package.json file, by running
 - `npm install`
 - it will install everything the project needs, in the `node_modules` folder, creating it if it's not existing already.
 - install a specific package by running: `npm install <package-name>`

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node Package Manager (npm)

- npm is the standard package manager for Node.js.
- It started as a way to download and manage dependencies of Node.js packages.
- Yarn and pnpm are alternatives to npm cli. You can check them out as well.
- Installing all dependencies
 - If a project has a package.json file, by running: `npm install`
 - it will install everything the project needs, in the `node_modules` folder, creating it if it's not existing already.
 - install a specific package by running: `npm install <package-name>`
 - `--save-dev` installs and adds the entry to the package.json
 - `--no-save` installs but does not add the entry to the package.json
 - `--save-optional` installs and adds the entry to the package.json file
 - `--no-optional` will prevent optional dependencies from being installed

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node Package Manager (global / local)

- **local packages** are installed in the directory where you run `npm install <package-name>`, and they are put in the `node_modules` folder under this directory
- **global packages** are all put in a single place in your system (exactly where depends on your setup), regardless of where you run `npm install -g <package-name>`
- A package **should be installed globally** when it provides an executable command that you run from the shell (CLI), and it's reused across projects.
- All projects have their own local version of a package, even if this might appear like a waste of resources, it's minimal compared to the possible negative consequences.
- You can also install executable commands locally and run them using `npm`, but some packages are just better installed globally.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node Package Manager (package.json)

- The `package.json` file is kind of a manifest for your project.
- It's a central repository of configuration for tools
- It's also where `npm` and `yarn` store the names and versions for all the installed packages.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node Package Manager (package.json)

- JSON file Structure
 - `version` indicates the current version
 - `name` sets the application/package name
 - `description` is a brief description of the app/package
 - `main` sets the entry point for the application
 - `private` if set to true prevents the app/package to be accidentally published on npm
 - `scripts` defines a set of node scripts you can run
 - `dependencies` sets a list of npm packages installed as dependencies
 - `devDependencies` sets a list of npm packages installed as development dependencies
 - `engines` sets which versions of Node.js this package/app works on
 - `browserslist` is used to tell which browsers (and their versions) you want to support

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Node Package Manager

- Update All Packages
- Install the `npm-check-updates` package globally:
 - `npm install -g npm-check-updates`
- Now run **npm-check-updates** to upgrade all version hints in `package.json`, allowing installation of the new major versions
 - `ncu -u`
- Finally, run a standard install
 - `npm install`

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Mobile-first paradigm

- responsive web design
- accommodates the screen size and other device attributes
- mobile-first, mean that the design an application first to work well on a mobile device and then move on to devices with larger screens.
- In Stylesheet, required Media queries where, for certain sized screens, the styles defined for mobile devices are overridden to make sense for devices with larger screens.

```
@media screen and (min-width: 600px) {
  /* For screens above 600px width */
}
@media screen and (min-width: 960px) {
  /* For screens above 960px width */
}
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Mobile-first paradigm

- At least target these device scenarios:
- **Small:** This includes iPhone 4.
- **Medium:** This can refer to tablet computers, or the larger smart phones.
- **Large:** This includes larger tablet computers, or the smaller desktop computers.
- **Extra-large:** This refers to larger desktop computers and other large screens.
- **Landscape/portrait:** You may want to create a distinction between landscape mode and portrait mode. Switching between the two of course changes viewport width, possibly pushing it past a breakpoint

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Using Twitter Bootstrap on the Notes application

- Setting it up
- `npm i bootstrap`
- `import`
`\./node_modules/bootstrap/dist/css/bootstrap.min.css'`
`;`
- `import`
`\./node_modules/bootstrap/dist/js/bootstrap.min.js'`;

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



CSS Flexbox and Grid CSS

- Grid vs. Flexbox
 - Grid is made for two-dimensional layout while Flexbox is for one. **Flexbox** can work on **either row or columns at a time**, but **Grids** can work on **both**.
 - Flexbox, gives you more flexibility while working on either element (row or column). HTML markup and CSS will be easy to manage in this type of scenario.
 - GRID gives you more flexibility to move around the blocks irrespective of your HTML markup.
 - Flex Direction allows developers to align elements vertically or horizontally while, In CSS Grid, auto-keyword functionality to automatically adjust columns or rows.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Synchronous vs Asynchronous

- `npm i fs -save`
- Every method in the **fs** module has synchronous as well as asynchronous forms.
- Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error.
- It is better to use an asynchronous method instead of a synchronous method, as the former never blocks a program during its execution, whereas the second one does.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Synchronous vs Asynchronous

```
var fs = require("fs");

// Asynchronous read
fs.readFile('input.txt', function (err, data) {
  if (err) {
    return console.error(err);
  }
  console.log("Asynchronous read: " + data.toString());
});

// Synchronous read
var data = fs.readFileSync('input.txt');
console.log("Synchronous read: " + data.toString());

console.log("Program Ended");
```

https://www.tutorialspoint.com/nodejs/nodejs_file_system.htm

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



File operations

- **r**: Open file for reading.
- **r+**: Open file for reading and writing
- **rs**: Open file for reading in synchronous mode.
- **w**: Open file for writing.
- **wx**: Like 'w' but fails if the path exists.
- **w+**: Open file for reading and writing.
- **a**: Open file for appending.
- **a+**: Open file for reading and appending.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Creating a Basic HTTP Server

- First create a folder to manage server
 - mkdir http-server
- Create a new file named server.js
- Write the following code in server.js


```
const http = require("http");
const host = 'localhost';
const port = 8000;
const requestListener = function (req, res) {
  res.writeHead(200);
  res.end("First HTTP server!");
};
const server = http.createServer(requestListener);
server.listen(port, host, () => {
  console.log("Server is running on http://${host}:${port}");
});
>node server.js
```

<https://www.digitalocean.com/community/tutorials/how-to-create-a-web-server-in-node-js-with-the-http-module>

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Returning Different Types of Content (HTML)

- Create a new file named `html.js`
- Write the following code in `server.js`

```
const http = require("http");

const host = 'localhost';
const port = 8000;

const requestListener = function (req, res) {
  res.setHeader("Content-Type", "text/html");
  res.writeHead(200);
  res.end('<html><body><h1>This is HTML</h1></body></html>');
};

const server = http.createServer(requestListener);
server.listen(port, host, () => {
  console.log('Server is running on http://$' + host + ':' + port);
});
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Returning Different Types of Content (JSON)

- Create a new file named `json.js`
- Write the following code in `server.js`

```
const http = require("http");

const host = 'localhost';
const port = 8000;

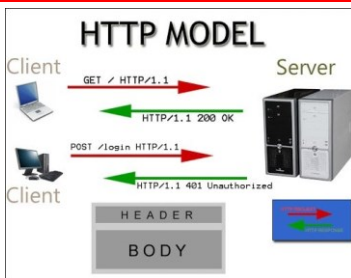
const requestListener = function (req, res) {
  res.setHeader("Content-Type", "application/json");
  res.writeHead(200);
  res.end(JSON.stringify({message: "This is a JSON response"}));
};

const server = http.createServer(requestListener);
server.listen(port, host, () => {
  console.log('Server is running on http://$' + host + ':' + port);
});
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



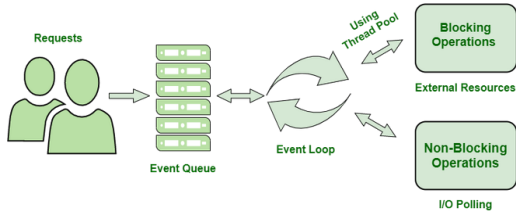
Web Application Architecture



<https://www.section.io/engineering-education/http-requests-nodes/>

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

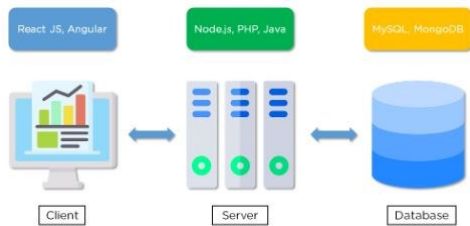
Web Application Architecture



<https://www.geeksforgeeks.org/node-js-web-application-architecture/#~:text=js%20Server%20Architecture%3A%20To%20manage,js%20Processing%20Model.>

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

Web Application Architecture



<https://www.simplilearn.com/understanding-node-js-architecture-article>

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita

Express Framework

- minimal and flexible Node.js web application framework
- provides a robust set of features to develop web and mobile applications
- It facilitates the rapid development of Node based Web applications.
- Allows to set up middlewares to respond to HTTP Requests.
- Defines a routing table which is used to perform different actions based on HTTP Method and URL.
- Installing Express
 - `npm install express --save`
- **body-parser** – This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.
- **cookie-parser** – Parse Cookie header and populate req.cookies with an object keyed by the cookie names.
 - `npm install body-parser --save`
 - `npm install cookie-parser --save`

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Example

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World');
});

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
});
```

https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Express Framework (Request & Response)

- Express application uses a callback function whose parameters are request and response objects.

```
app.get('/', function (req, res) {
  // --
});
```

- Request Object** – The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.
- Response Object** – The response object represents the HTTP response that an Express app sends when it gets an HTTP request.
- Examples already discussed in practical lab sessions.

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Express Framework (Cookies)

- Cookies are
 - small files/data that are sent to client with a server request stored on the client side.
- Every time the user loads the website back, this cookie is sent with the request. This helps us keep track of the user's actions.
- The following are the numerous uses of the HTTP Cookies –
 - Session management
 - Personalization(Recommendation systems)
 - User tracking.

```
var cookieParser = require('cookie-parser');
app.use(cookieParser());

app.get('/', function(req, res){
  res.cookie('name', 'express').send('cookie set'); //Sets name = express
});
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Express Framework (Cookies)

- Adding Cookies with Expiration Time

```
//Expires after 360000 ms from the time it is set.
res.cookie(name, 'value', {expire: 360000 + Date.now()});
```

- Deleting Existing Cookies

```
app.get('/clear_cookie_foo', function(req, res){
  res.clearCookie('foo');
  res.send('cookie foo cleared');
});
```

https://www.tutorialspoint.com/expressjs/expressjs_sessions.htm

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Express Framework (Session)

- HTTP is stateless; in order to associate a request to any other request, you need a way to store user data between HTTP requests.

```
npm install --save express-session
```

```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');

var app = express();

app.use(cookieParser());
app.use(session({secret: "Shh, its a secret!}));

app.get('/', function(req, res){
  if(req.session.page_views){
    req.session.page_views++;
    res.send("You visited this page " + req.session.page_views + " times");
  } else {
    req.session.page_views = 1;
    res.send("Welcome to this page for the first time!");
  }
});
app.listen(3000);
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita



Express Framework (Session)

```
var express = require('express');
var cookieParser = require('cookie-parser');
var session = require('express-session');

var app = express();

app.use(cookieParser());
app.use(session({secret: "Shh, its a secret!}));

app.get('/', function(req, res){
  if(req.session.page_views){
    req.session.page_views++;
    res.send("You visited this page " + req.session.page_views + " times");
  } else {
    req.session.page_views = 1;
    res.send("Welcome to this page for the first time!");
  }
});
app.listen(3000);
```

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Dr. Arpita